Structuring Rust Programs

Sam Whited

JID: sam@samwhited.com

2018–04–18

The Problem

Filesystem

1. `src/`

Filesystem

2. `src/mymodule.rs`

Filesystem

3. `src/mymodule/mod.rs`

Filesystem

3. `src/mymodule/{mod.rs, other.rs}`

Keywords

1. `extern crate`

Keywords

2. `mod`

Keywords

2. `pub mod`

Keywords

2. `pub mod mymod {}`

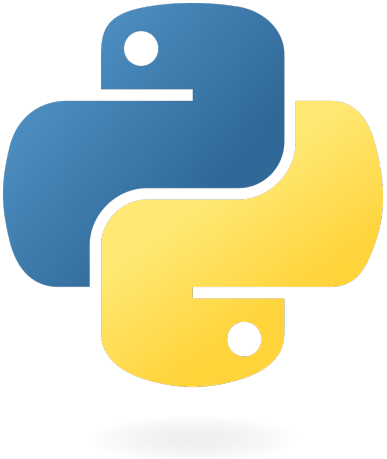
Keywords

3. `use`

Keywords

3. `pub use`

This seems familiar...



(s/mod.rs/__init__.py/)

Applications

`main.rs`

What goes in main?

`main.rs`

1. Keep it small.

`main.rs`

1. Keep it small.
2. Keep it self contained.

`main.rs`

1. Keep it small.
2. Keep it self contained.

But it's already full of `extern crate`...

`main.rs`

"Process-level logic"

main.rs

1. Signal handling

`main.rs`

1. Signal handling
2. Error handling and exit status

`main.rs`

1. Signal handling
2. Error handling and exit status
3. Flag / arg parsing

main.rs

1. Signal handling
2. Error handling and exit status
3. Flag / arg parsing
4. Construction of application wide deps

`main.rs`

1. Signal handling
2. Error handling and exit status
3. Flag / arg parsing
4. Construction of application wide deps
    1. Logging

main.rs

1. Signal handling
2. Error handling and exit status
3. Flag / arg parsing
4. Construction of application wide deps
    1. Logging
    2. Metrics

main.rs

1. Signal handling
2. Error handling and exit status
3. Flag / arg parsing
4. Construction of application wide deps
    1. Logging
    2. Metrics
    3. ...

main.rs

1. Signal handling
2. Error handling and exit status
3. Flag / arg parsing
4. Construction of application wide deps
    1. Logging
    2. Metrics
    3. ...
5. Initialization and handoff to the application level

main.rs (GTK)

```
fn main() {
    gtk::init().expect(translate!("failed_to_start_gtk"));
    match app::App::new() {
        Err(e) => {
            eprintln!("Error registering app: {:?}", e);
            // Nothing else should ever call process:exit
            process::exit(1);
        }
        Ok(app) => {
            app.run(&args().collect::<Vec<_>>());
        }
    }
}
```

main.rs (web)

```
fn main() {
    let tmpls = Rc::new(compile_templates!("templates/**/*"));
    let logger = myloggersetup(io::stdout());

    // Metrics
    let metrics = prometheus::Registry::new();
    let counter_opts = Opts::new("startup", "Shows app startup");
    let counter = Counter::with_opts(counter_opts).unwrap();
    metrics.register(Box::new(counter.clone())).unwrap();
    counter.inc();

    app::Server::new(tmpls, logger, metrics).serve("127.0.0.1:3000")
}
```

Modules

Modules

`widgets.rs`

`widgets/mod.rs`

File Modules

Save the trees, prefer single file modules.

Block Modules

Block Modules

Just don't.

Block Modules

Just don't*.

*Except maybe for tests, at your disgression. Be consistent.

Tree Modules

1. Avoid until you need more than one file

Tree Modules

1. Avoid until you need more than one file
2. Avoid putting logic in `mod.rs`

Tree Modules

1. Avoid until you need more than one file
2. Avoid putting logic in `mod.rs` (but avoid "util" modules more)

Tree Modules

1. Avoid until you need more than one file
2. Avoid putting logic in `mod.rs` (but avoid "util" modules more)
3. Avoid "tiny module syndrome"

Tree Modules

1. Avoid until you need more than one file
2. Avoid putting logic in `mod.rs` (but avoid "util" modules more)
3. Avoid "tiny module syndrome" (but avoid "util" modules more)

Tree Modules

1. Avoid until you need more than one file
2. Avoid putting logic in `mod.rs` (but avoid "util" modules more)
3. Avoid "tiny module syndrome" (but avoid "util" modules more)
4. Avoid "util" modules

Tree Modules

1. Avoid until you need more than one file
2. Avoid putting logic in `mod.rs` (but avoid "util" modules more)
3. Avoid "tiny module syndrome" (but avoid "util" modules more)
4. Avoid "util" modules

mod.rs

```
mod about;
mod app_menu;
mod chat;
mod header_bar;
mod login;
mod roster;

pub use self::about::about_dialog;
pub use self::app_menu::app_menu;
pub use self::chat::Chat;
pub use self::header_bar::header_bar;
pub use self::login::Login;
```

Most Importantly...

Consider the import path.

Common Problems

Consider the import path.

Stutter

```
reader::Reader::new_reader()
```

Stutter

`reader::Reader::new_~~reader()~~`

Stutter

```
reader::Reader::new()
```

Stutter

```
reader::Reader::new()
```

Stutter

```
io::Reader::new()
```

Import Ambiguity

```
main() {
    parse();
}
```

Import Ambiguity

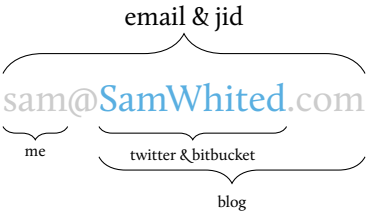
```
use config::parse;

main() {
    parse();
}
```

Import Ambiguity

```
use config;

main() {
    config::parse();
}
```


email & jid
sam@SamWhited.com
me
twitter & bitbucket
blog